Chap. 05
Deep Learning & HPC

中山浩太郎
松尾研究室

Speed Matters

Why speed matters

"Faster is better" applies almost everywhere, not just in the tech domain.

by Joshua-Michéle Ross | @jmichele | April 26, 2011

処理速度が2倍だったら...

- 2倍のデータを処理
- 2倍のパラメータを探索
- 2倍の実験を回せる
- 計算機を専有する時間が1/2に
- より多くの研究者で計算機を共有できる
- 研究に要する時間が短く
- 研究プロセスを早くできる
- 問題を発見するのに要する時間が短く
- AWSに払うお金が半分に

悲しい事件その1

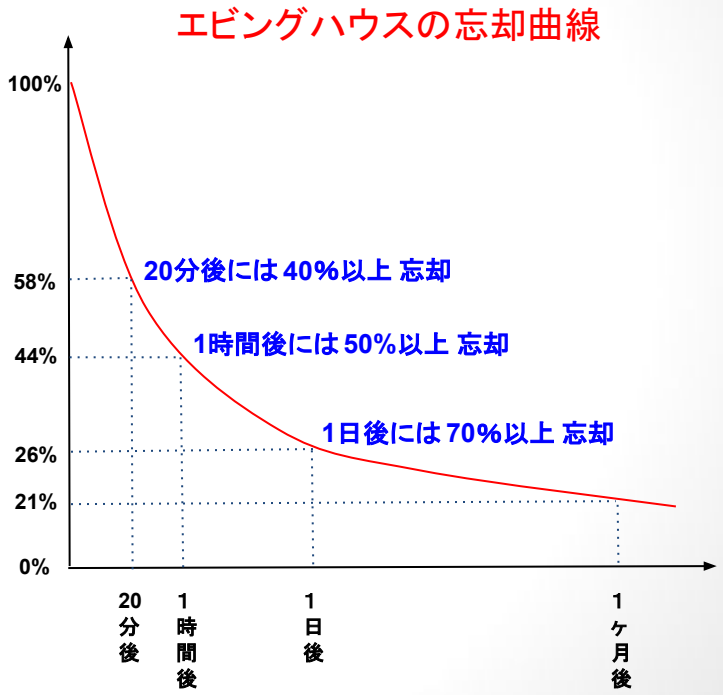
「君のプロセスが10日もGPUを専有してるんだけど」

↓

「忘れてました」

エビングハウスの忘却曲線

- パラメータのことを思い出すためのオーバーヘッド
- 数日も経つとそもそも実験を回していることも忘れる
- １週間も経つと既に研究の興味は別のことに移っている
- byobuの功罪



悲しい事件その2

「松尾先生のクレジットカードにAmazonから30万円の請求が来てるけど誰か知ってる？」

↓

学生「AWSって高いんですね」

良い表現学習のためには
大量のデータが必要

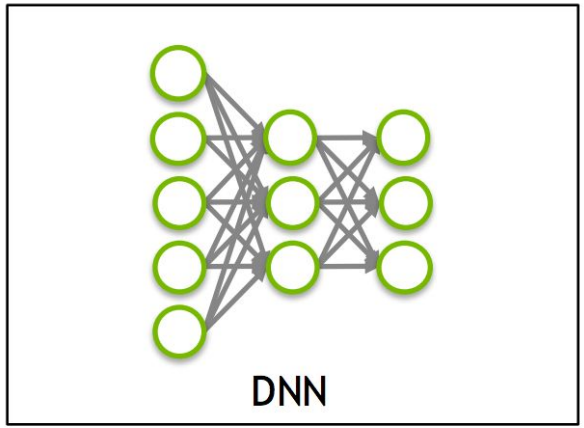
データセットとサイズ

DataSet	Records	Compressed	Memory
CIFAR10 (32 x 32)	60K	160MB	235MB
Caltech101 (JPEG)	10K	126MB	2,000MB
STL-10 (96 x 96)	100k	2,500MB	2,600MB
MSCOCO 2014 (Train, JPEG)	**82K**	**13GB**	**N.M**
ImageNet (Train)	**??**	**138GB**	**N.M**
ImageNet (Test)	**??**	**13GB**	**N.M**

一般的な環境でのメモリ量比較
松尾研究室: 64/128 GB, AWS GPU: 15 / 60 GB
とてもメインメモリに乗らない
（環境に応じてCropping / 解像度を落とす、分散処理等）

例えば医療画像

The Big Bang in Machine Learning



"Google's AI engine also reflects how the world of computer hardware is changing. (It) depends on machines equipped with GPUs... And it depends on these chips more than the larger tech universe realizes."

WIRED

from NVIDIA - GPU Tech Conference 2016

Agenda

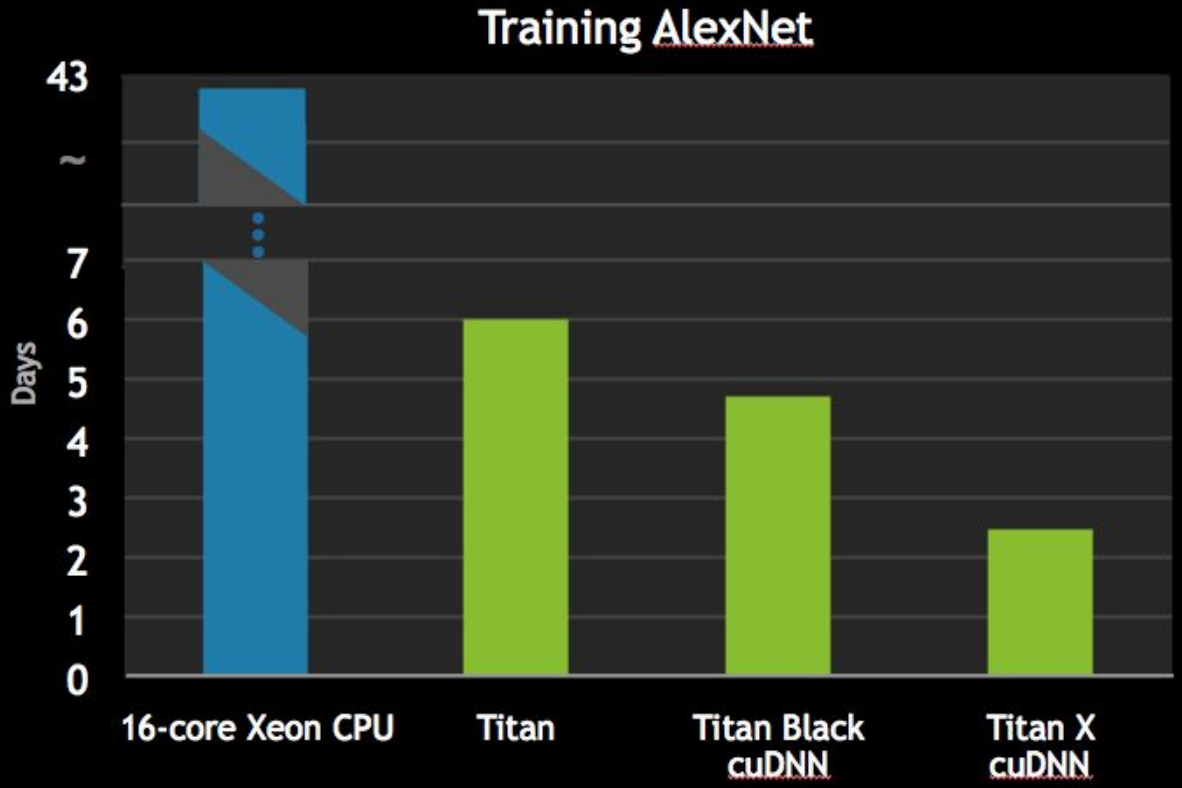
- HPCとDeep Learning
- 大規模データとストレージ
- 並列計算・マルチGPU
- 実践テクニック
- まとめ

Agenda

- HPCとDeep Learning
- 大規模データとストレージ
- 並列計算・マルチGPU
- 実践テクニック
- まとめ

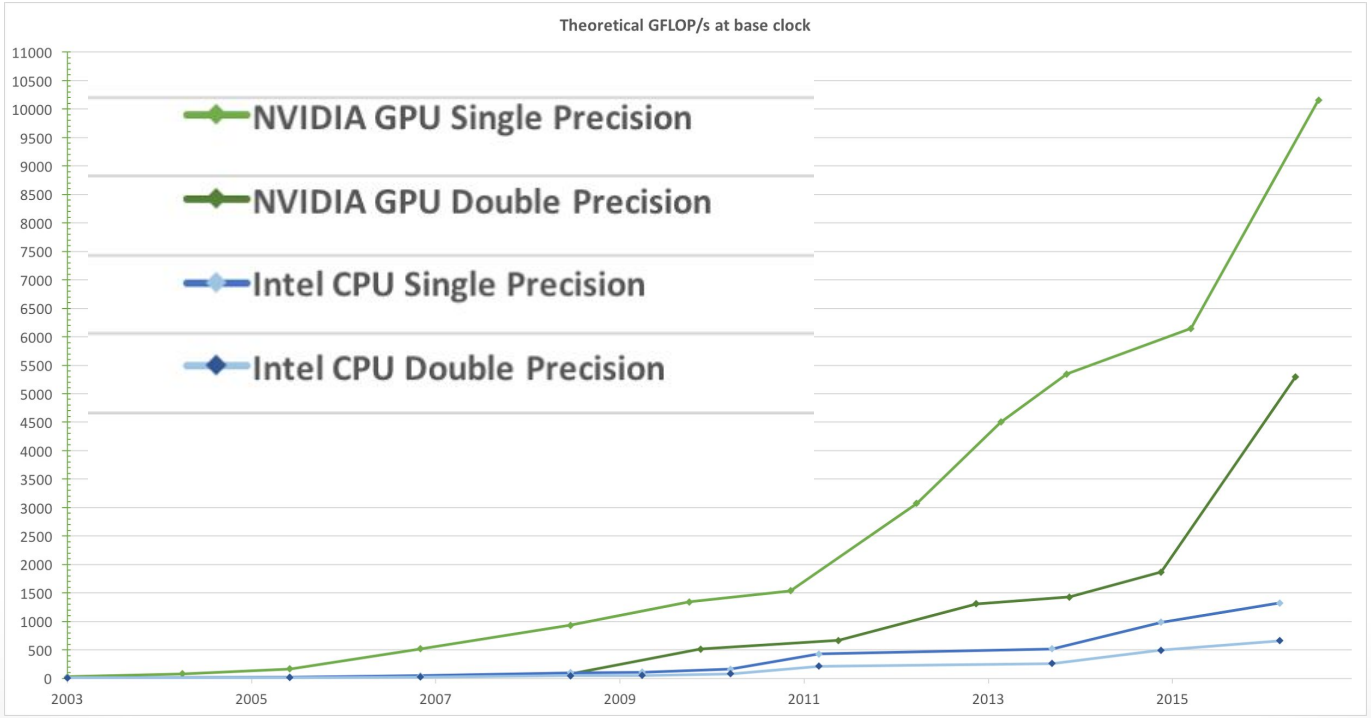
Deep LearningとGPU

TITAN X FOR DEEP LEARNING



from NVIDIA

GPUとムーアの法則



docs.nvidia.com

Deep Learning
ライブラリ一覧

ライブラリ	主な特徴	利用方法	GPU対応
Caffe	・高速な動作・モデルや実装を共有する コミュニティが充実 ・画像処理のモデルが充実	・Protobufファイルでモデル作成 ・Python / Matlab インタフェース ・C++言語でのモデル作成	◎
Torch7	・高度なパッケージ管理機能 ・モダンな設計思想 ・シンプルかつ強力なモデル設計機能	・Lua言語でのモデル作成	◎
Keras	・Theanoベース ・モダンな設計思想 ・LSTM、RNNなどにも対応	・Python	◎
Chainer	・Pythonベース、Numpyとの高い親和性 ・導入が容易, define-by-run	・Python ・Caffeの設定ファイル読み込み	◎
TensorFlow	・Pythonベース、Numpyとの高い親和性 ・自動微分	・Python	◎
Pytorch	・Pythonベース、Numpyとの高い親和性 ・define-by-run	・Python	◎

そもそもGPUって何？

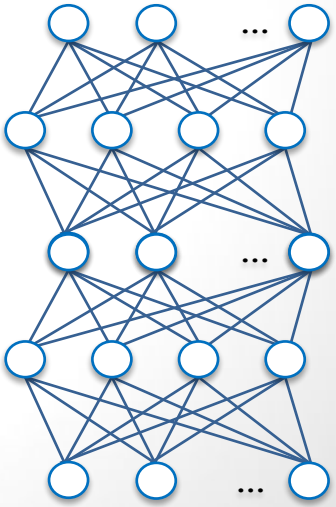
GPU?

- グラフィック（3D等）のレンダリングに利用
- Nvidia Geforce GTX 780 Ti
- 2,880コア
  5,040 GFLOPS
- Intel Xeon E5-4650
  8コア
  172.8 GFLOPS
- 高度な並列性と高いメモリ帯域幅

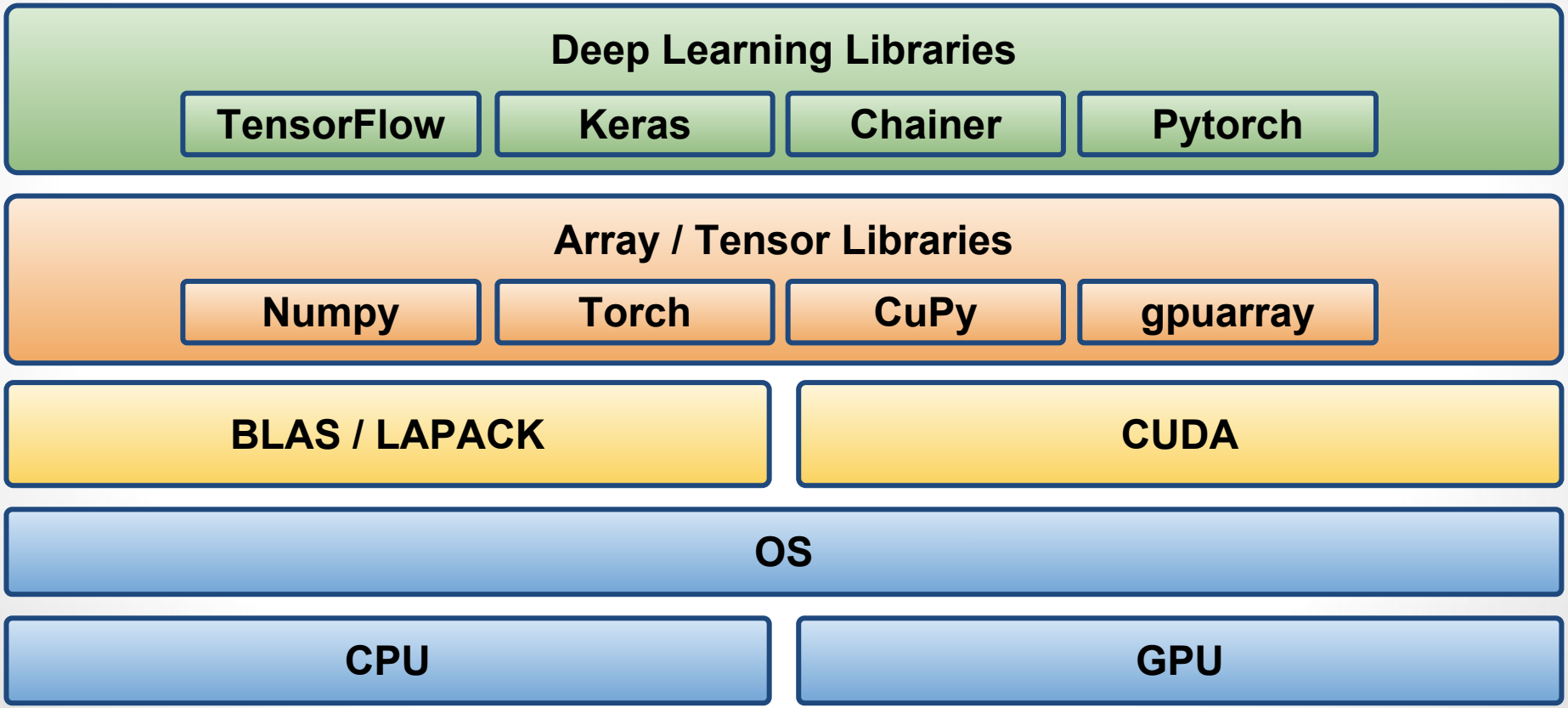
Source: Nvidia

Deep Learning (NN) と GPU

- 高度な並列性と高いメモリ帯域幅
- ニューラルネットワークの計算特性と同じ
  - 大量のパラメータ、活性値、勾配のためのバッファ
  - 単純な行列計算・行列積が計算の中心
  - GPUの並列計算特性に合致
- GPGPUの普及
- 2005年にGPGPUでMLPを3x高速化
- その後すぐに畳み込みネットワークにも適用



Python系フレームワーク



例）松尾研究室の環境

- 仮想化サーバでCPU・メモリ・GPUを共有
- すべてマルチGPU環境
- 教員・学生でリソース共有
- カンファレンス締め切り前には争奪戦

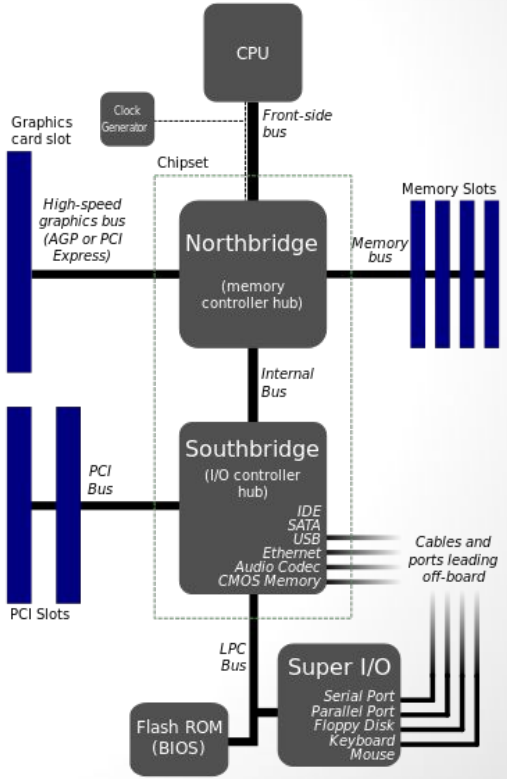
GPU Type	Cores	MEM	#
TITANX	3,000	12GB	12 (+2)
TITANX (Pascal)	3,584	12GB	8 (+4)
TESLA K40	2,880	12GB	4
GeForce 980GTX	2,000	6GB	2
TITAN Black	2,880	6GB	3 (-1)

マルチGPU環境の必要性と問題

- １台のマシンに複数のGPU
- GPUサーバの管理コスト低減
- 仮想化による共有
- マルチGPUによる並列計算・勾配計算
- 問題
  - 使用メモリなども気を使う
  - 締め切り前の争奪戦

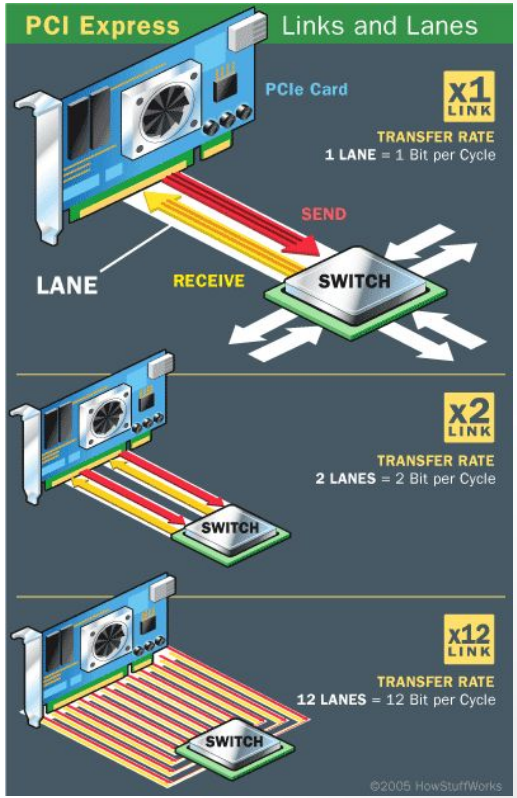
PCアーキテクチャ, CPU, GPU

- Northbridge
  - CPU, Main Memory, PCIe
  - Frontside bus (FSB) – CPU
  - 非常に高速
- Southbridge
  - USB, SATA, Ethernet, etc.
  - I/O controller hub
  - Northbridgeと比べて低速



PCIe and Lanes

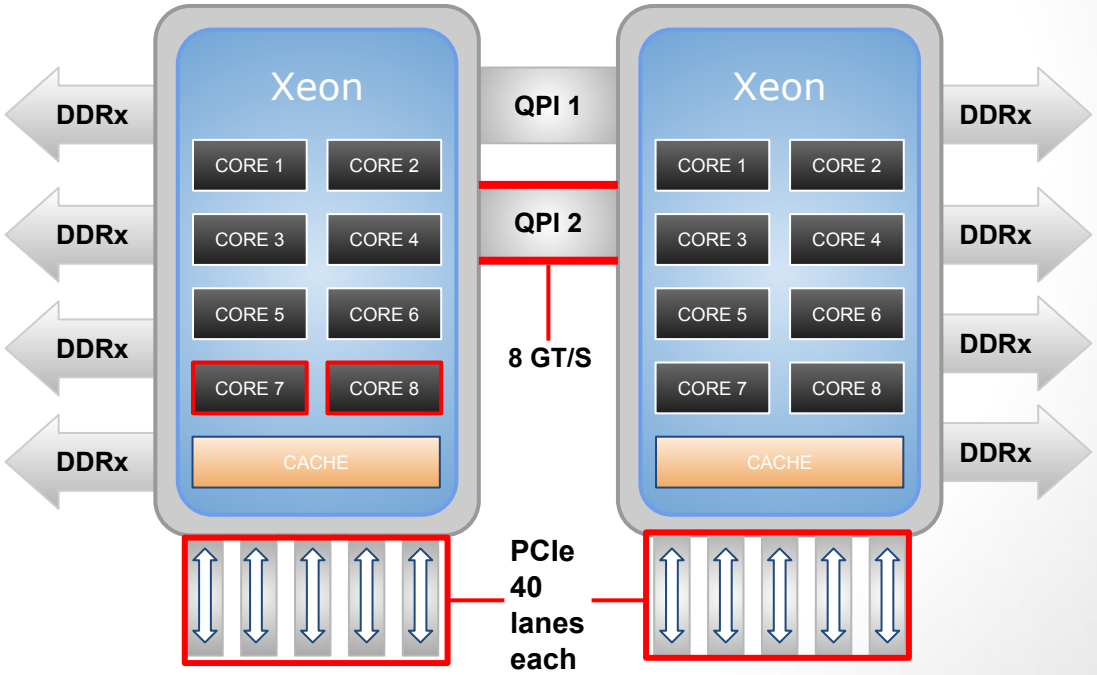
- 1x PCIe 3.0 (1レーン)
  - 500MB / 1 Way
- 16x PCIe
  - 16x send, 16x receive
  - 16GB / Sec
- 40レーンの壁
  - サーバグレードXeonでも
    2つしかGPUを搭載できない



howstuffworks.com

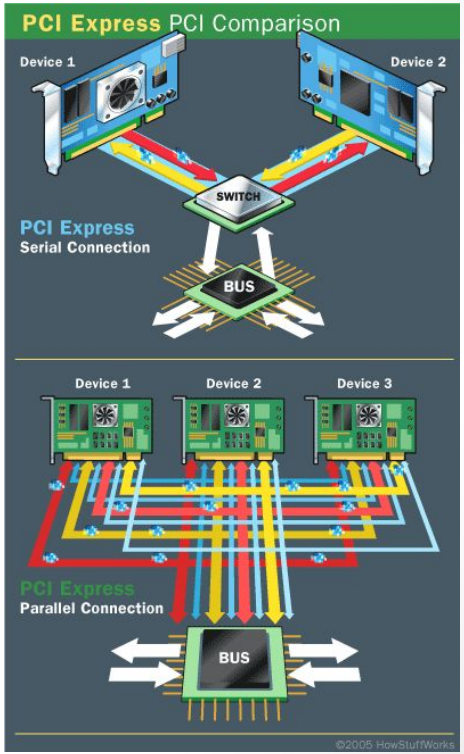
DP環境

- Xeon X 2
- 合計80レーン
- 16xのGPUを
  それぞれ2台
- 合計4台まで
- QPI:
  Quick Path
  Interconnect



PCIe and Lanes

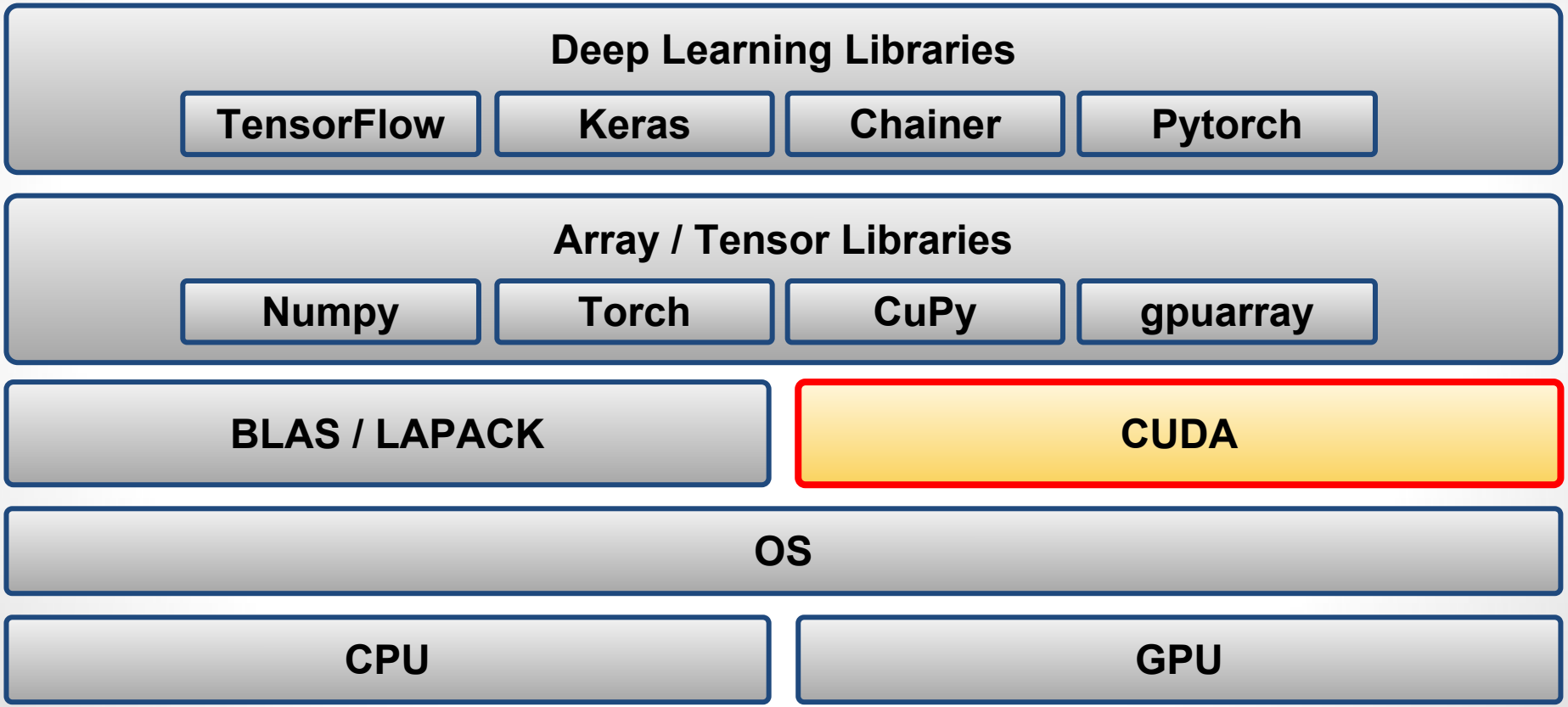
- 特殊なマザーボード
- PCI Expressのバスに
  スイッチ機構
- 16x GPUを8台搭載可能に



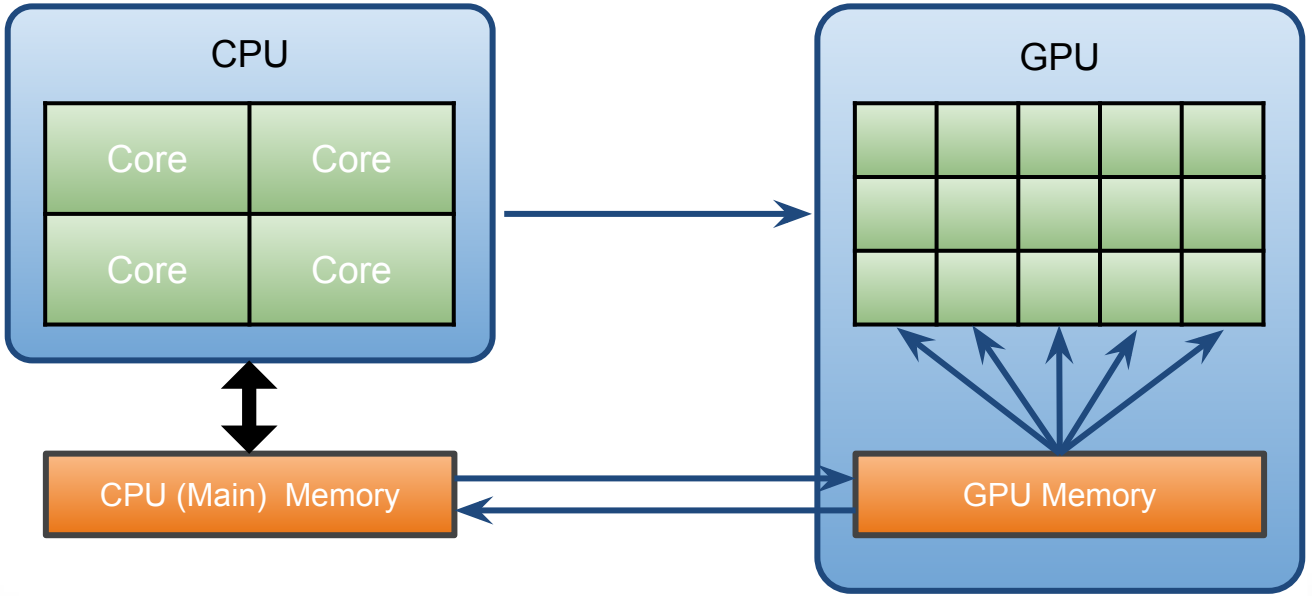
howstuffworks.com

GPUとCUDAカーネル

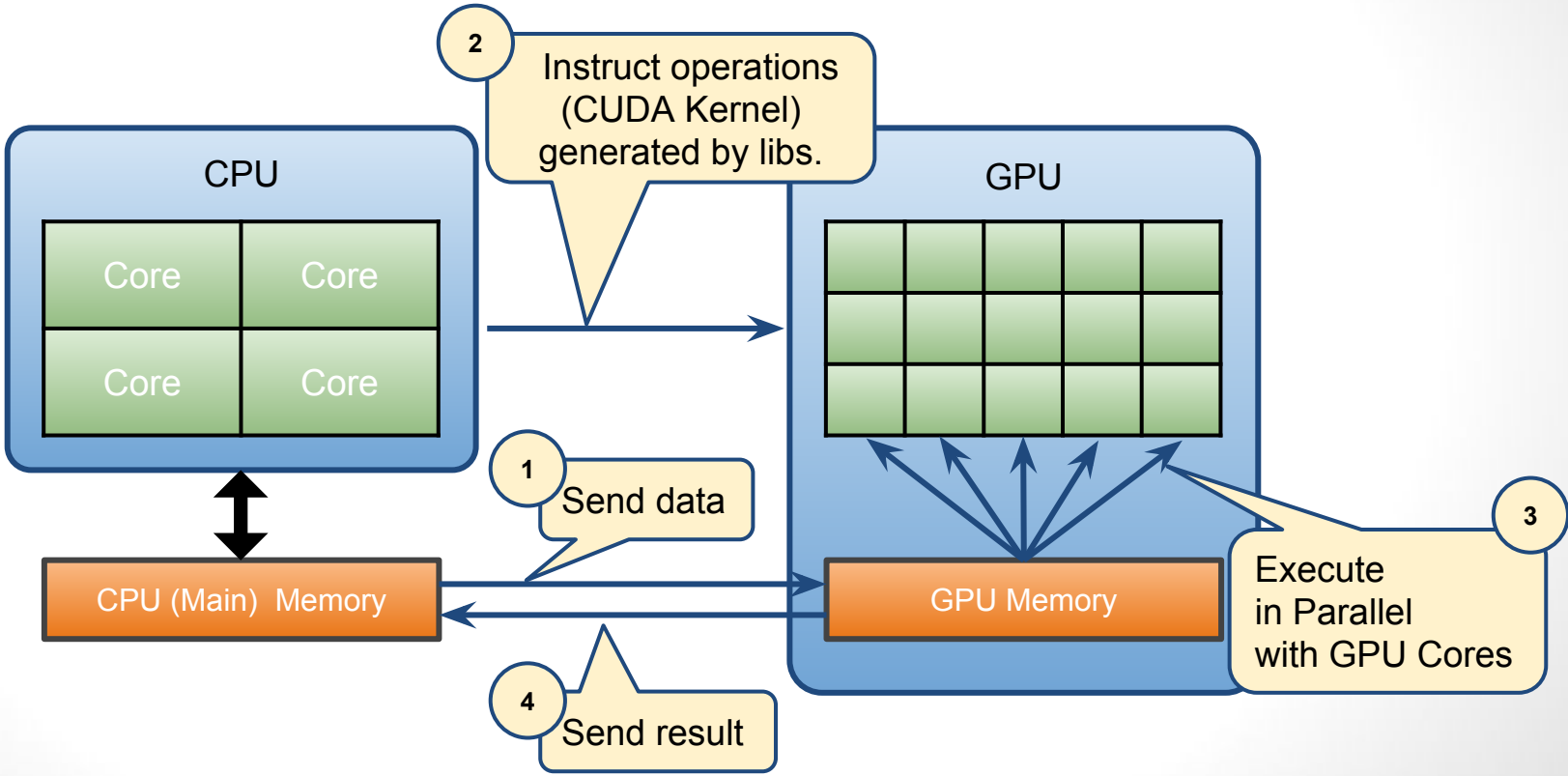
Python系フレームワーク



GPU – CPU 処理フロー



GPU – CPU 処理フロー



ここまでのまとめ

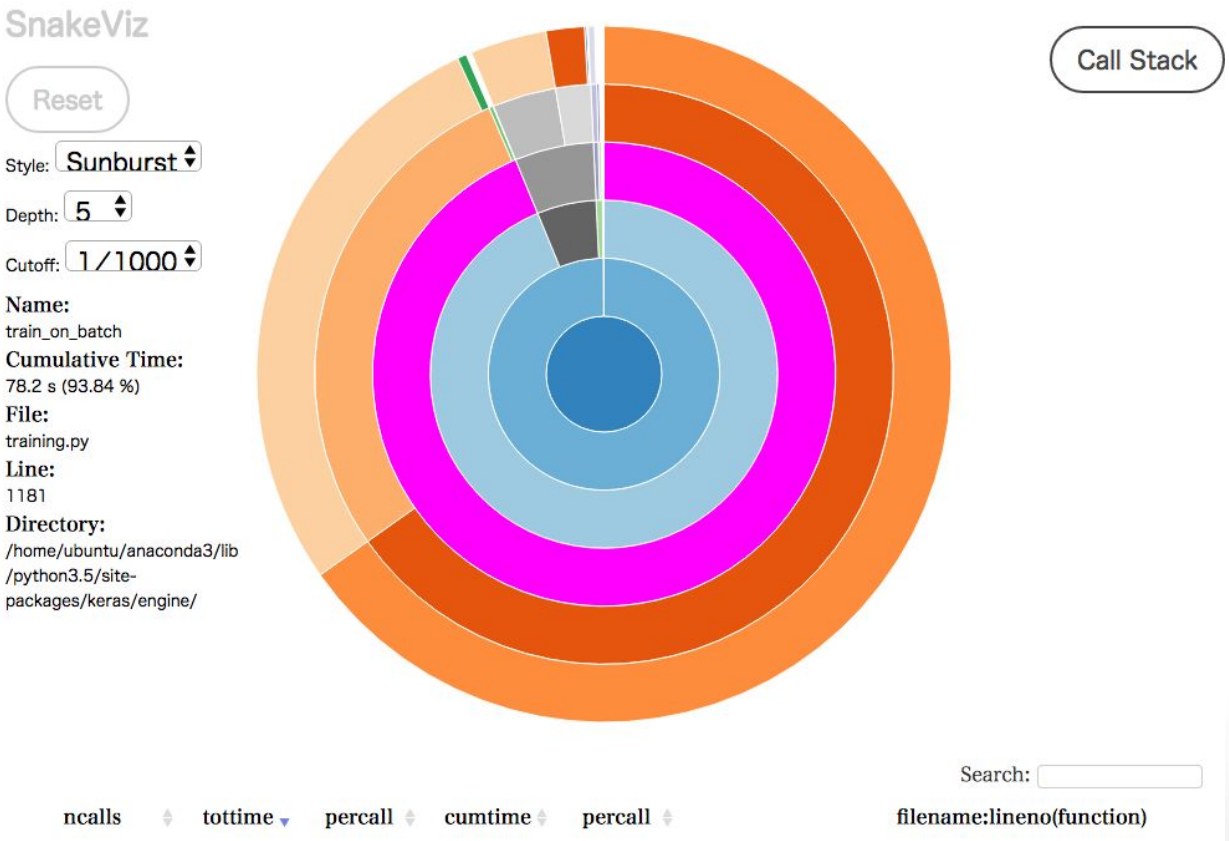
- Deep LearningにGPUは必須
- ミドルウェアがCUDAのコードを自動生成
- テンソル処理・畳み込み計算の高速化
- マルチGPU環境
- GPU<->CPUレーンを意識しないと
  パフォーマンス低下
- 処理フローやボトルネックを知ることが大事

Profiler

Profiler

- Profilerとは
  - プログラムのパフォーマンスを計測
  - どの関数呼び出しがどれくらいの実行時間を使っているか
  - 可視化のツールも充実
  - プログラムのボトルネック発見には必須
- PythonのProfiler（cProfile）
- CUDAのProfiler

CPU Profiler



Memory Profiler

```
(3)ubuntu@k-nakayama_cuda:~/tmp$ python -m memory_profiler load_cifar10.py
Using Theano backend.
Using gpu device 0: Tesla K40c (CNMeM is enabled with initial size: 95.0% of memory, cuDNN not avai
(50000, 32, 32, 3)
(10000, 32, 32, 3)
Filename: load_cifar10.py

Line #    Mem usage    Increment   Line Contents
================================================
     3  218.586 MiB    0.000 MiB   @profile
     4                              def load_data():
     5  453.270 MiB  234.684 MiB       (X_train, y_train), (X_test, y_test) = cifar10.load_data()
     6  453.270 MiB    0.000 MiB       print(X_train.shape)
     7  453.270 MiB    0.000 MiB       print(X_test.shape)
```

Exercise

SnakeViz

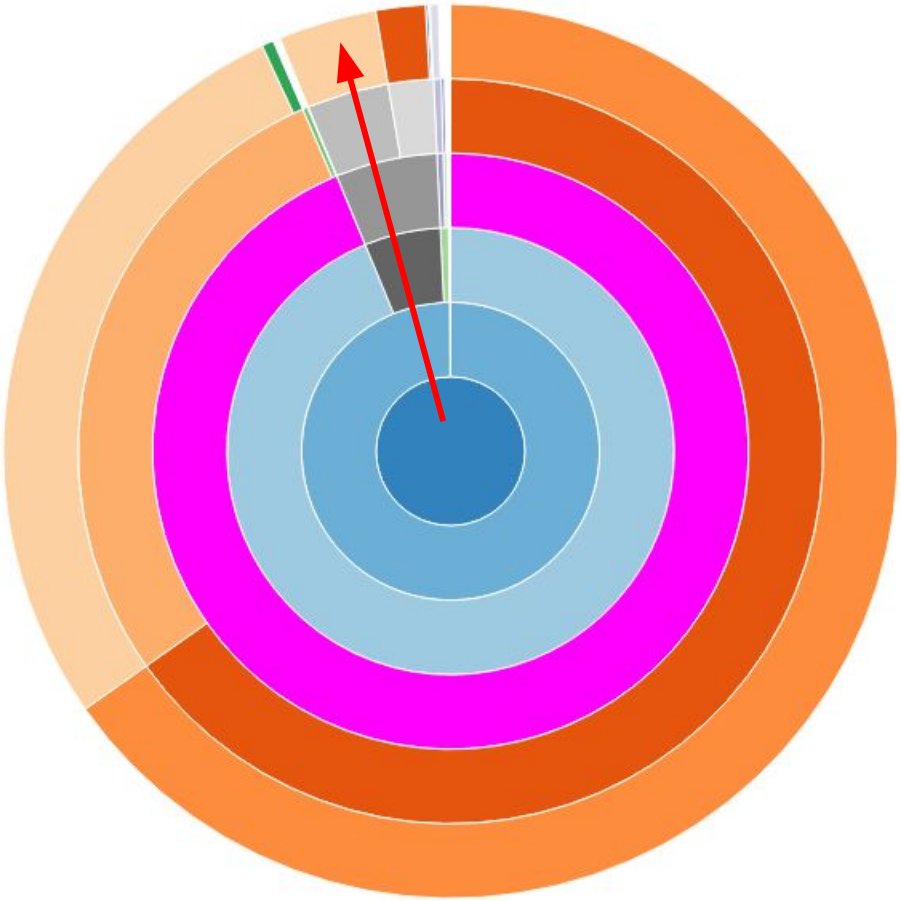
Reset

Style: Sunburst

Depth: 5

Cutoff: 1/1000

Name:
train_on_batch
Cumulative Time:
78.2 s (93.84 %)
File:
training.py
Line:
1181
Directory:
/home/ubuntu/anaconda3/lib/python3.5/site-packages/keras/engine/



Call Stack

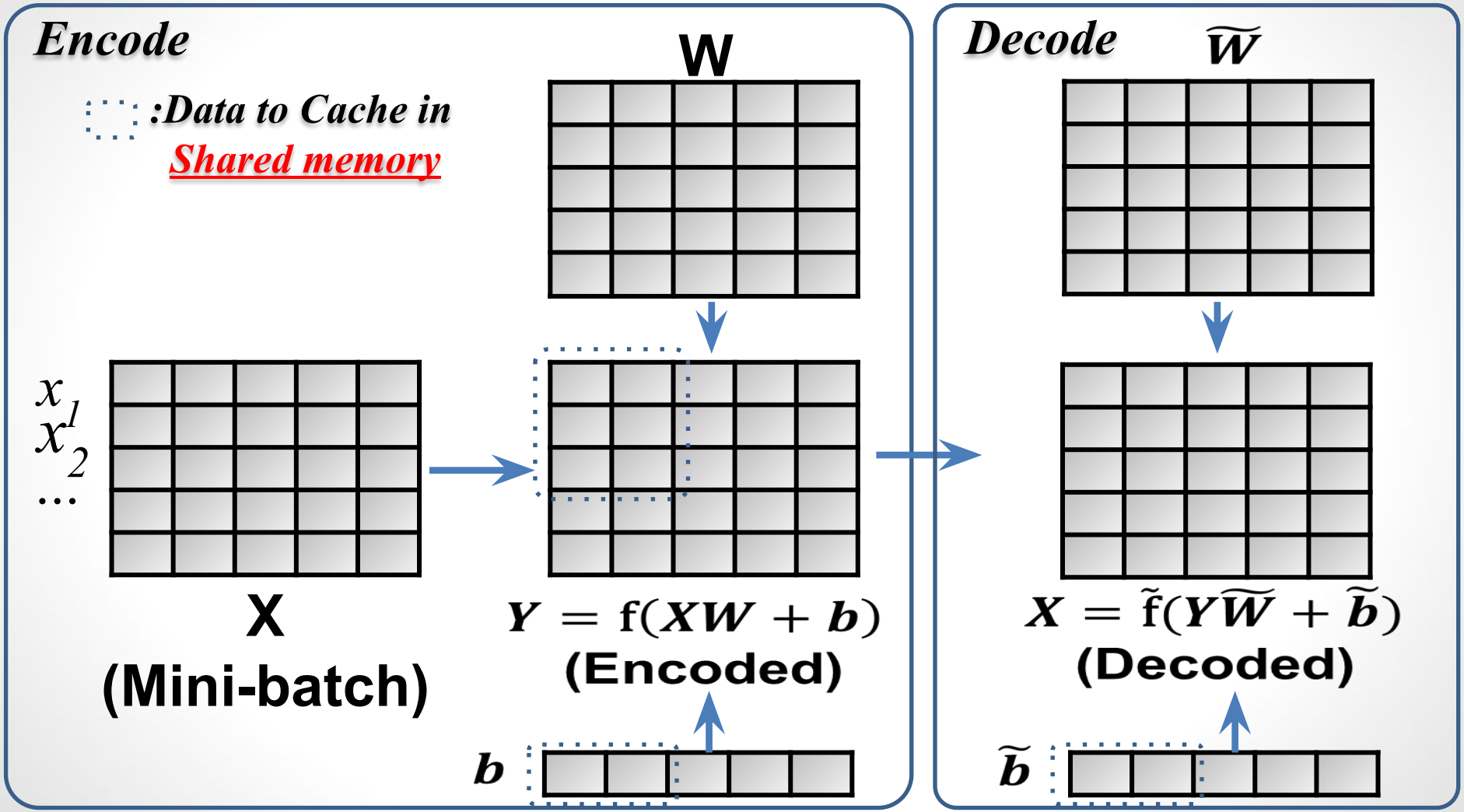
Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)

GPUで
どんな処理も高速になるか？
↓
高速化の効果が高い処理は限定

行列積と畳込み計算

Minibatch SGD


Encode
:Data to Cache in
Shared memory
W
x_1
x_2
...
X
(Mini-batch)
$Y = f(XW + b)$
(Encoded)
b
Decode
$\widetilde{W}$
$X = \tilde{f}(Y\widetilde{W} + \widetilde{b})$
(Decoded)
$\widetilde{b}$

Mini-batch SGD

Batch-SGD

全部

X	y
X	y
X	y
X	y
X	y
X	y
X	y
X	y
X	y

SGD

1つだけ利用

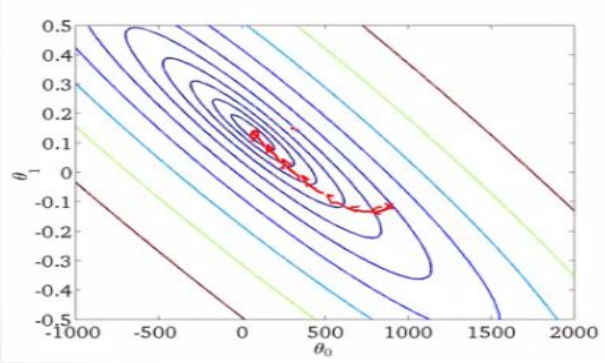
X	y

Minibatch SGD

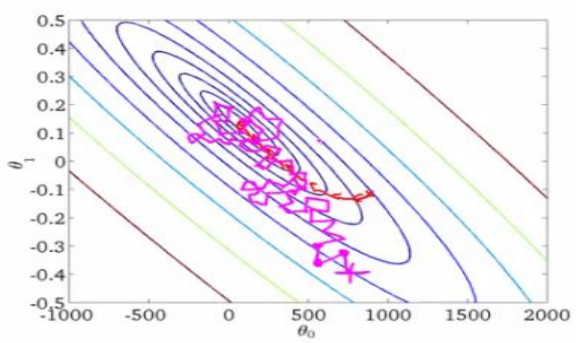
X	y
X	y
X	y

Mini-batch SGD

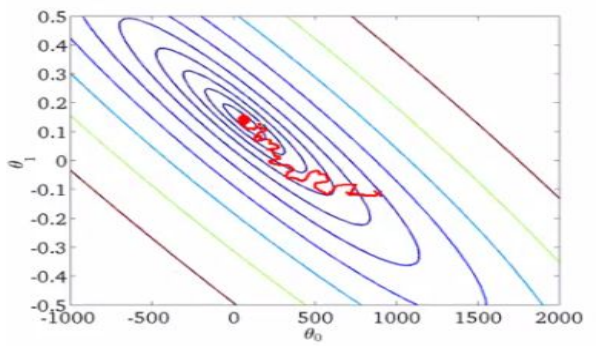
Batch-SGD
全部



SGD
1つだけ利用



Minibatch SGD



Andrew Ng.資料より

高速化TIPS

Illustration: Fabien Tencé - ankivil.com

高速化TIPS

経験上、30~40%高速化
最近は入れてないと挙動が怪しいフレームワークもある

経験上、10%～20%程度高速化
GPUメモリを最大限確保
メモリ領域をバッファ

Illustration: Fabien Tencé - ankivil.com

松尾研究室の計算機環境

- **参考:**
  http://weblab.t.u-tokyo.ac.jp/hpc/
- **前提:**
  - **学生はサーバの使い方が荒い**
  - **研究者も荒い(sudo rm -r / 事件)**
  - **すぐに環境を壊す**
  - **Deep Learning関係の実装は依存関係が多い**
  - **Deep Learning関係の技術革新はおろそしいスピードで進んでいる**
- **環境の使い捨てが可能な仮想化技術が必須**

GPU Type	Cores	MEM	#
TITANX	3,000	12GB	12 (+2)
TITANX (Pascal)	3,584	12GB	8 (+4)
TESLA K40	2,880	12GB	4
GeForce 980GTX	2,000	6GB	2
TITAN Black	2,880	6GB	3 (-1)

GPUで
どんな処理も高速になるか？
↓
条件分岐や複雑な処理はX

Agenda

- HPCとDeep Learning
- 大規模データとストレージ
- 並列計算・マルチGPU
- 実践テクニック
- まとめ

データセットとサイズ

DataSet	Records	Compressed	Memory
CIFAR10 (32 x 32)	60K	160MB	235MB
Caltech101 (JPEG)	10K	126MB	2,000MB
STL-10 (96 x 96)	100k	2,500MB	2,600MB
MSCOCO 2014 (Train, JPEG)	**82K**	**13GB**	**N.M**
ImageNet (Train)	**??**	**138GB**	**N.M**
ImageNet (Test)	**??**	**13GB**	**N.M**

一般的な環境でのメモリ量比較
松尾研究室: 64/128 GB, AWS GPU: 15 / 60 GB
とてもメインメモリに乗らない
（環境に応じてCropping / 解像度を落とす、分散処理等）

ストレージ

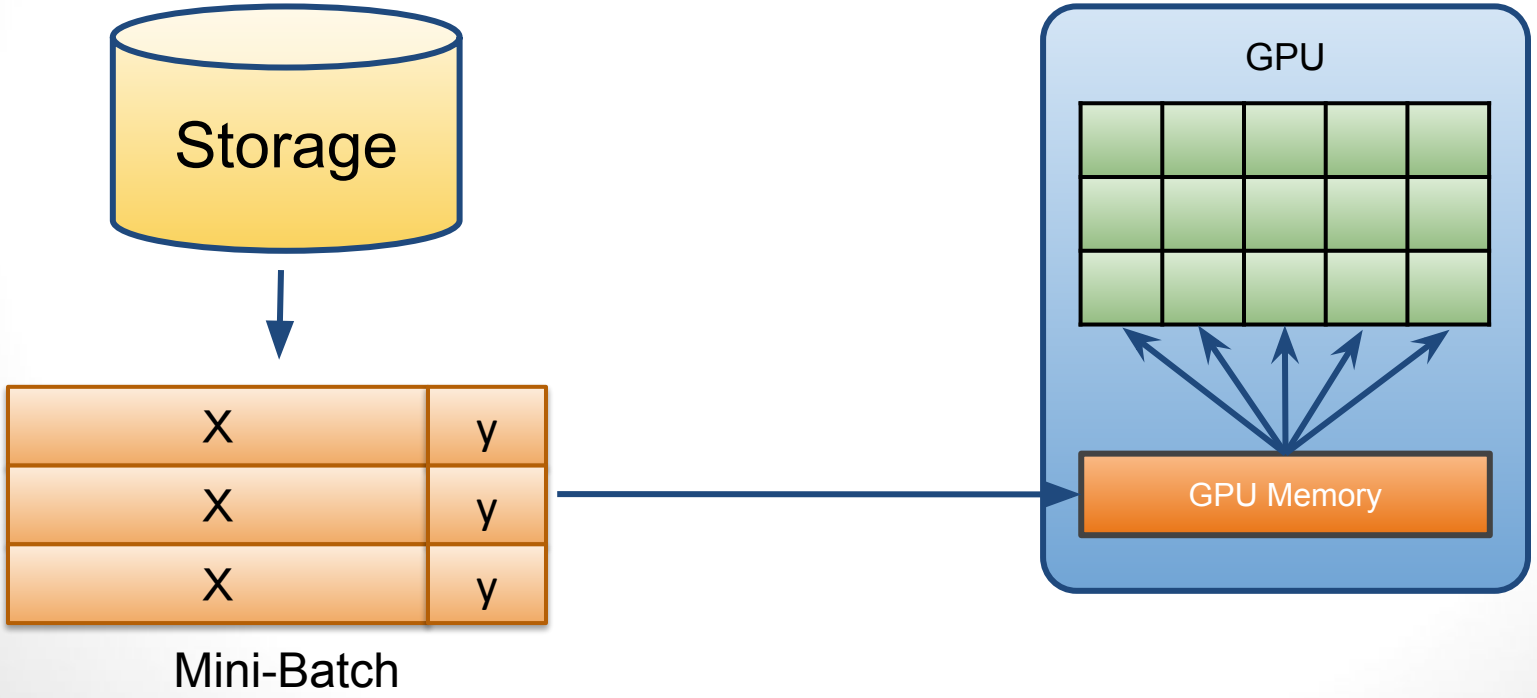
Deep Learningとストレージ
DB系

- スケーラビリティ・クエリなどで利点
- メモリよりは当然低速
- LMDB
  - 高速
  - Caffeでよく利用される
  - 並列読み込みやOSキャッシュの利用で高速化
- HDF5
  - 柔軟なクエリ
  - Numpy Arrayがそのまま突っ込める
  - ネットワークのシリアライズなどでも利用

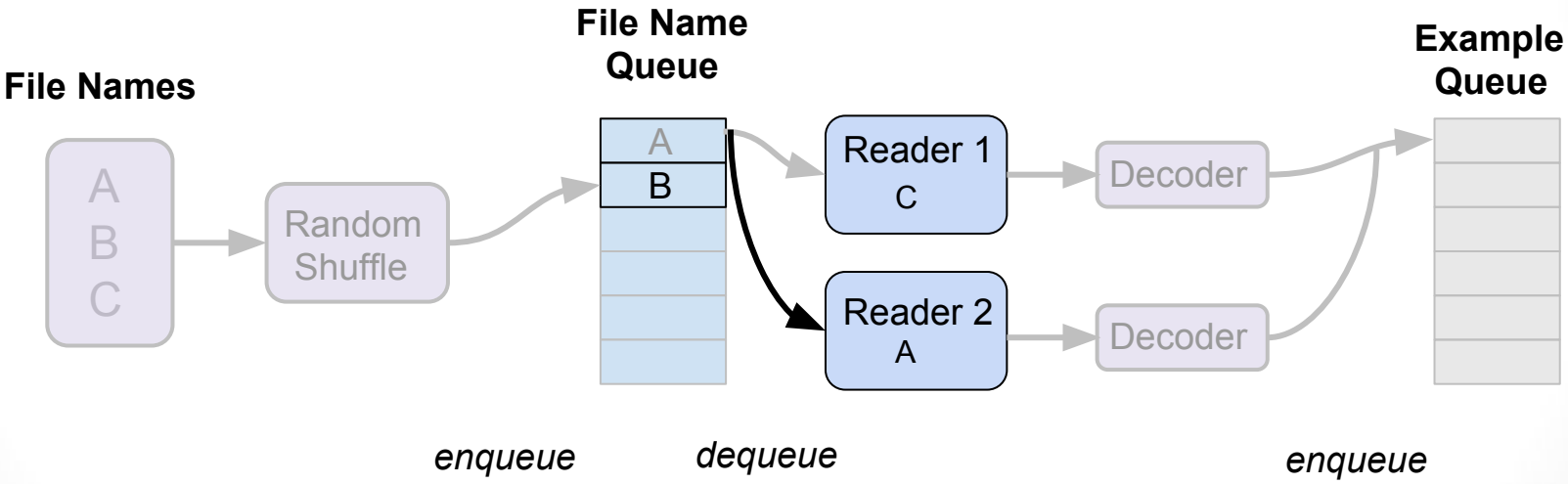
ファイルシステム系

- ライブラリ付属のデータジェネレータ
  - Keras, TensorFlow, ...
  - Caltech形式（サブディレクトリ=クラス）
- Spark DataFrame + TF

Deep Learningと ストレージ



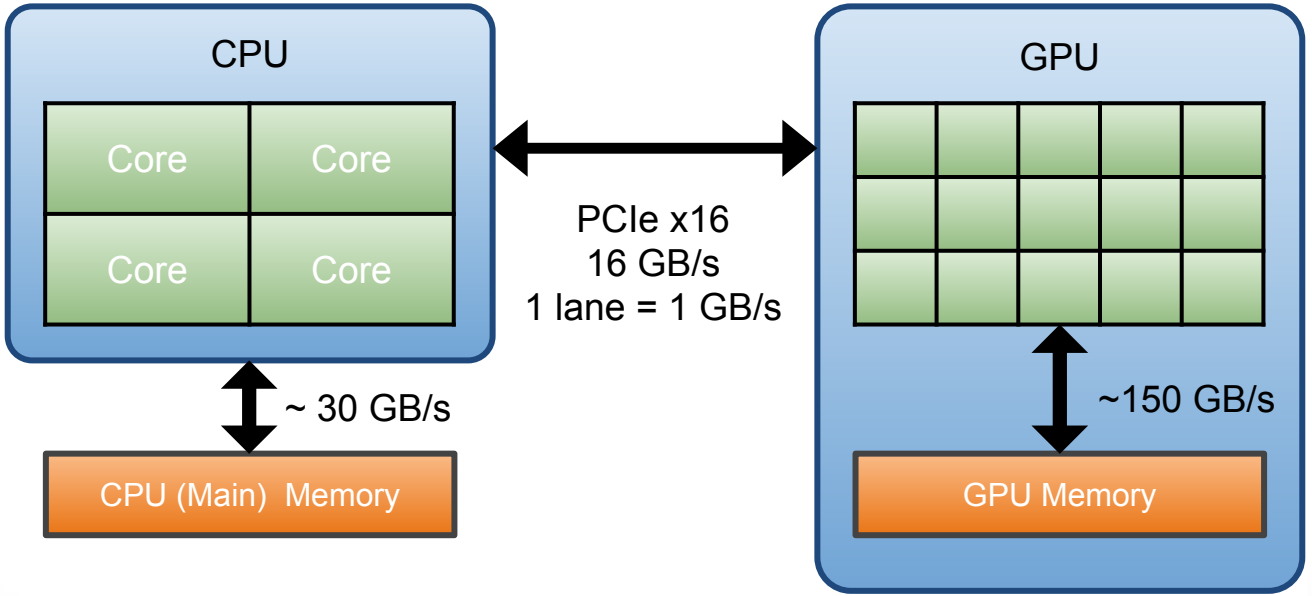
並列読み込み



NFS/SSH << File IO << Main Mem << GPU Mem
時間のかかる読み込み処理を並列化・キュー管理

Exercise

CPUとGPU – 通信と速度

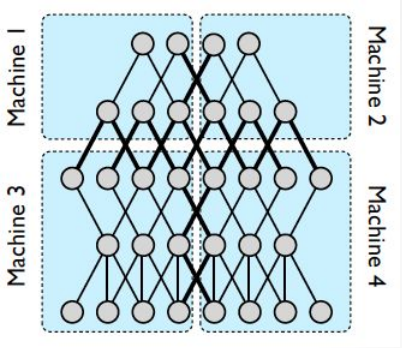


Agenda

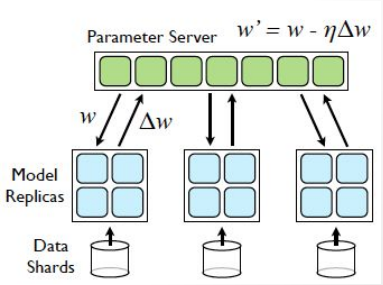
- HPCとDeep Learning
- 大規模データとストレージ
- 並列計算・マルチGPU
- 実践テクニック
- まとめ

Deep Learningと並列計算

- パラメータサーチ
  - 複数のプロセス(マシン)に
    個別のパラメータの組み合わせを割り当てる
  - LR, Dropout Rate, # of neurons
  - グリッドサーチ, ランダムサーチ, BO
- データ並列
  - 複数GPU(マシン)に
    それぞれ個別のデータ(ミニバッチ)を割り当てる
  - 一つのモデルを共有
  - 推論に適用するのは非常に容易
- モデル並列
  - 一つのモデルを複数のマシンで分割計算
  - モデル毎に作り込みが必要



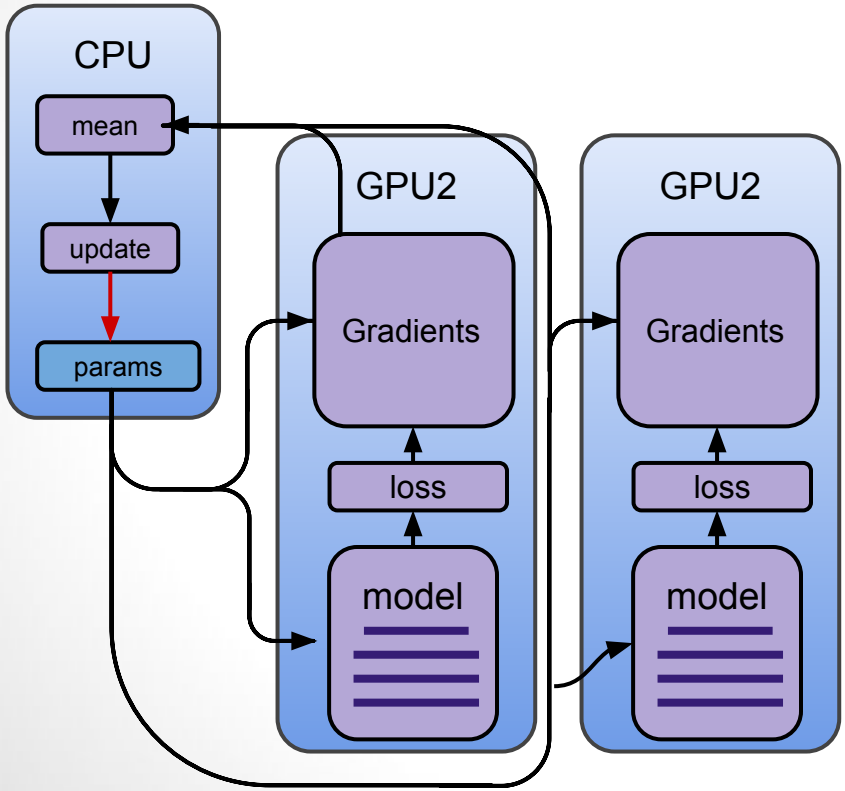
モデル並列

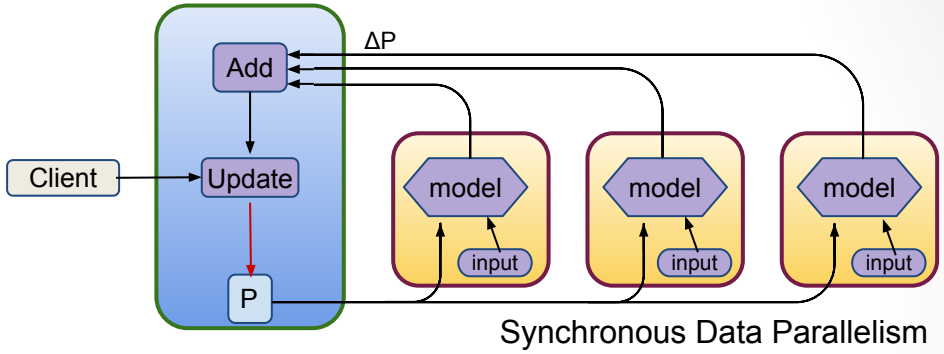


データ並列

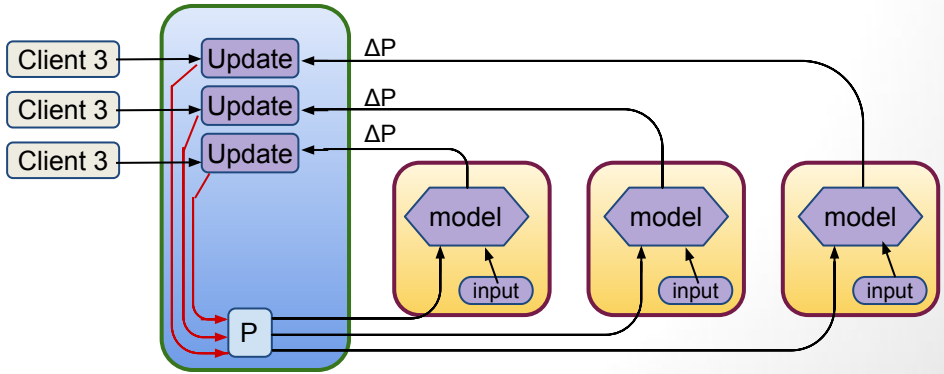
J. Dean, et al., Large Scale Distributed Deep Networks, NIPS 2012

マルチGPUと分散処理





Synchronous Data Parallelism



Asynchronous Data Parallelism

非同期確率的勾配降下法（Asynchronous SGD）

Hogwild!
(Feng Niu, et al. 2011)

- パラメータ読み込み：非同期（わかる）
- パラメータ書き込み：非同期（！）
- コアの一部がお互いの進行を上書き
- 各グラジエント降下ステップの平均的な改善量は減少
- スパースな問題の場合にはステップの生産速度の増加
- 全体としては学習プロセスの高速化を持たらす

```
import torch.multiprocessing as mp
from model import MyModel

def train(model):
    # Construct data_loader, optimizer, etc.
    for data, labels in data_loader:
        optimizer.zero_grad()
        loss_fn(model(data), labels).backward()
        optimizer.step()  # This will update the shared parameters

if __name__ == '__main__':
    num_processes = 4
    model = MyModel()
    # NOTE: this is required for the ``fork`` method to work
    model.share_memory()
    processes = []
    for rank in range(num_processes):
        p = mp.Process(target=train, args=(model,))
        p.start()
        processes.append(p)
    for p in processes:
      p.join()
```

Downpour SGD
Sandblaster L-BFGS

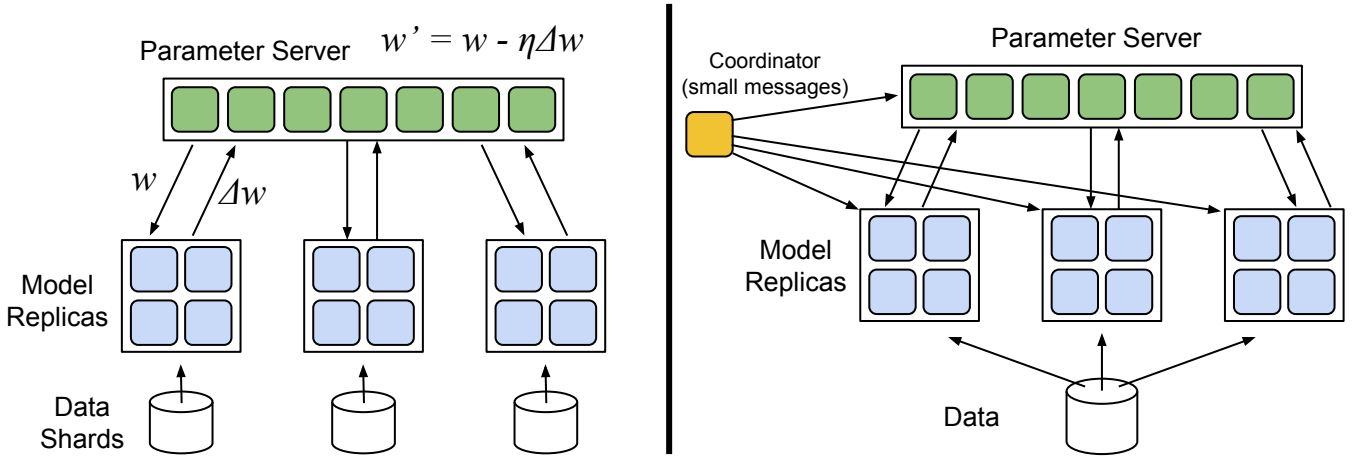


Figure 2: Left: Downpour SGD. Model replicas asynchronously fetch parameters w and push gradients Δw to the parameter server. Right: Sandblaster L-BFGS. A single 'coordinator' sends small messages to replicas and the parameter server to orchestrate batch optimization.

J. Dean, et al., Large Scale Distributed Deep Networks, NIPS 2012

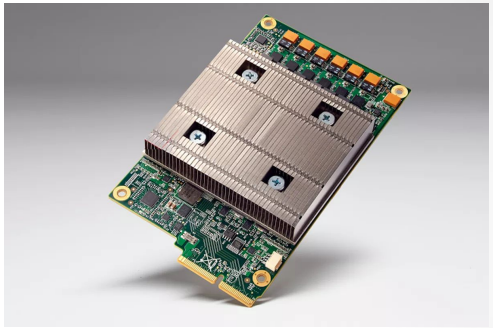
その後

- “Going Deeper with Convolutions,” (GoogLeNet), CVPR 2015
- “Batch Normalization”
- AlphaGoのポリシーネットワークの学習
- “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” Google White Paper, 2015

さらに高速の世界へ

専用ハードウェア

- ハードの種類
  - GPU
  - Application-Specific Integrated Circuit(ASIC)
  - FPGA
- Tensor Processing Unit (TPU)
  - Google開発のASIC
  - Google Street View、AlphaGoで利用



TPU
(from pcworld.com)

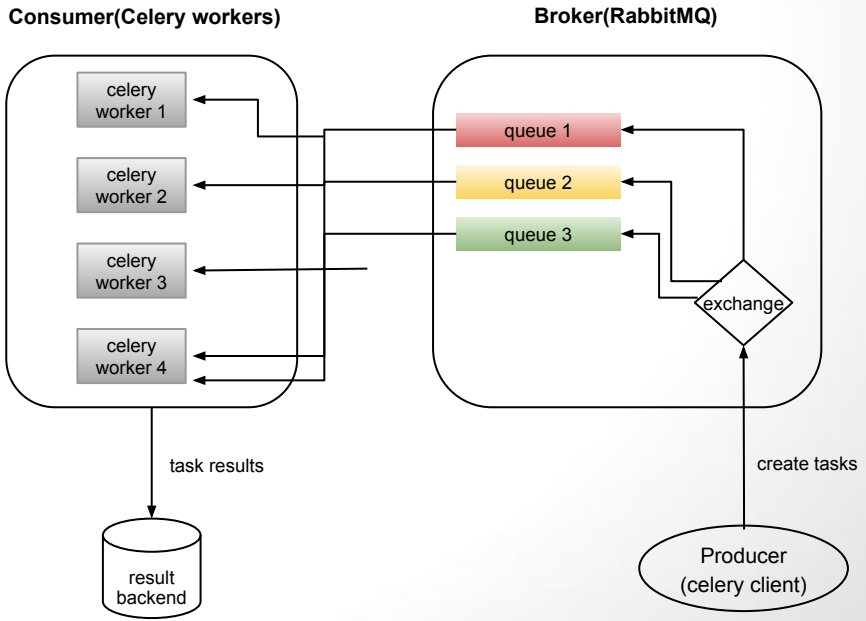
Agenda

- HPCとDeep Learning
- 大規模データとストレージ
- 並列計算・マルチGPU
- 実践テクニック
- まとめ

Celery

Celery

- 非同期分散ジョブ/タスクキュー
- 簡単にスケール
- アプリサーバとワーカーを分離
- Webサーバの計算部分分離



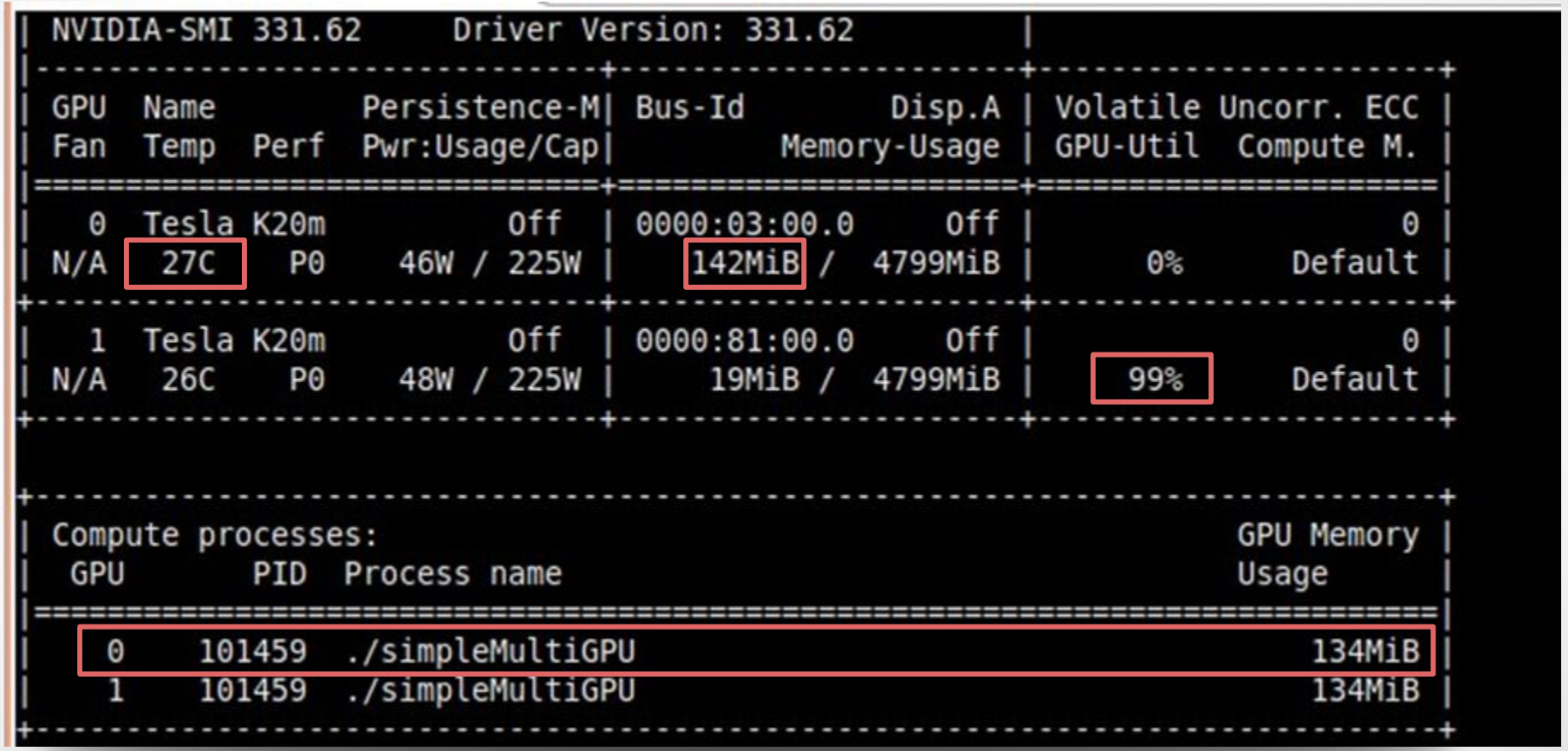
FacialVAE

http://fvae.ail.tokyo

M. Suzuki | WEBLAB

GPUツール

nvidia-smi



```
| NVIDIA-SMI 331.62     Driver Version: 331.62                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K20m          Off  | 0000:03:00.0     Off |                    0 |
| N/A   27C    P0    46W / 225W |    142MiB /  4799MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K20m          Off  | 0000:81:00.0     Off |                    0 |
| N/A   26C    P0    48W / 225W |     19MiB /  4799MiB |     99%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Compute processes:                                               GPU Memory |
|  GPU       PID  Process name                                     Usage      |
|=============================================================================|
|    0    101459  ./simpleMultiGPU                                    134MiB |
|    1    101459  ./simpleMultiGPU                                    134MiB |
+-----------------------------------------------------------------------------+
```

その他便利なツール

- Microway/gpu-burn
- nvprof
- cuda/samples
  - matrixMul
  - bandwidthTest
  - deviceQuery …

```
[root@node3 ~]# ./gpu_burn -d 60
GPU 0: Tesla K20m (UUID: GPU-1247410f-4b9e-81f7-bb2e-09b1cb1ae60a)
GPU 1: Tesla K20m (UUID: GPU-41a862d3-d7d8-d087-4c73-6aae177a79c4)
Initialized device 1 with 5119 MB of memory (5025 MB available, using 4523 MB of it), using DOUBLES
Initialized device 0 with 5119 MB of memory (5025 MB available, using 4523 MB of it), using DOUBLES
11.7%  proc: 2K/2K err: 0/0 tmp: 35C/39C
        Summary at:   Thu May 26 09:53:42 EST 2016

23.3%  proc: 5K/6K err: 0/0 tmp: 37C/41C
        Summary at:   Thu May 26 09:53:49 EST 2016

35.0%  proc: 8K/9K err: 0/0 tmp: 40C/44C
        Summary at:   Thu May 26 09:53:56 EST 2016

46.7%  proc: 11K/12K err: 0/0 tmp: 41C/45C
        Summary at:   Thu May 26 09:54:03 EST 2016

58.3%  proc: 14K/15K err: 0/0 tmp: 42C/46C
        Summary at:   Thu May 26 09:54:10 EST 2016

68.3%  proc: 17K/18K err: 0/0 tmp: 44C/48C
        Summary at:   Thu May 26 09:54:16 EST 2016

81.7%  proc: 20K/21K err: 0/0 tmp: 44C/49C
        Summary at:   Thu May 26 09:54:24 EST 2016

91.7%  proc: 23K/23K err: 0/0 tmp: 45C/50C
        Summary at:   Thu May 26 09:54:30 EST 2016

100.0%  proc: 26K/27K err: 0/0 tmp: 46C/51C
Killing processes.. done

Tested 2 GPUs:
        GPU 0: OK
        GPU 1: OK
[root@node3 ~]#
```

Agenda

- HPCとDeep Learning
- 大規模データとデータベース
- 並列計算・マルチGPU
- 実践テクニック
- まとめ

まとめ

- 良い表現学習 → 大規模データセット
- 各リソース特性と処理の流れを知ることでパフォーマンスを最大化できる
  - CPU、メモリ、ストレージ、GPU
- プロファイル
- ストレージ、データベース
- マルチGPU、並列処理